Theodor H. Nelson 8480 Fredericksburg #138 San Antonio TX 78229 512/ 692-7346

1

About the author.

Ted Nelson, a "top-down idealist," has been designing interactive systems for personal computers since 1960. Most of his writings are about one set of ideas, now implemented as the Xanadu(tm) Hypertext System, a storage module soon to be available for over-the-phone experimentation. It is further explained in his book Literary Machines (available from Project Xanadu, 8480 Fredericksburg, Suite 138, San Antonio TX 78229, for $15).

TYRF
d17

18 Aug 86

→ d17a, according to tangle.

THE TYRANNY OF THE FILE

Theodor H. Nelson

The fish does not see the water. And we computer people, even the savants and whizzes and frontiersmen, generally fail to see the most oppressive and devastating aspect of our working lives. I refer to the FILE. And the conceptual structure of the storage methods we must deal with. Chaotic and conceptually fragmented, the world of computer files is an enormous barrier to the clean systems of tomorrow.

We tell beginners it MUST BE THIS WAY. But I believe there has to be a fundamental redesign of the way we use computers, undoing this harm

Theodor H. Nelson 8480 Fredericksburg #138 San Antonio TX 78229 512/ 692-7346

<u>About the author.</u>

Ted Nelson, a "top-down idealist," has been designing interactive systems for personal computers since 1960. Most of his writings are about one set of ideas, now implemented as the Xanadu(tm) Hypertext System, a storage module soon to be available for over-the-phone experimentation. It is further explained in his book <u>Literary Machines</u> (available from Project Xanadu, 8480 Fredericksburg, Suite 138, San Antonio TX 78229, for $15).

TYRF
dif

19 Aug 86

THE TYRANNY OF THE FILE

Theodor H. Nelson

The fish does not see the water. And we computer people, even the savants and whizzes and frontiersmen, generally fail to see the most oppressive and devastating aspect of our working lives. I refer to the FILE. And the conceptual structure of the storage methods we must deal with. Chaotic and conceptually fragmented, the world of computer files is an enormous barrier to the clean systems of tomorrow.

We tell beginners it MUST BE THIS WAY. But I believe there has to be a fundamental redesign, of the way we use computers, undoing this harm and difficulty. A clean new interface to our worlds of information could sweep away a vast amount of the difficulty that people have with computers.

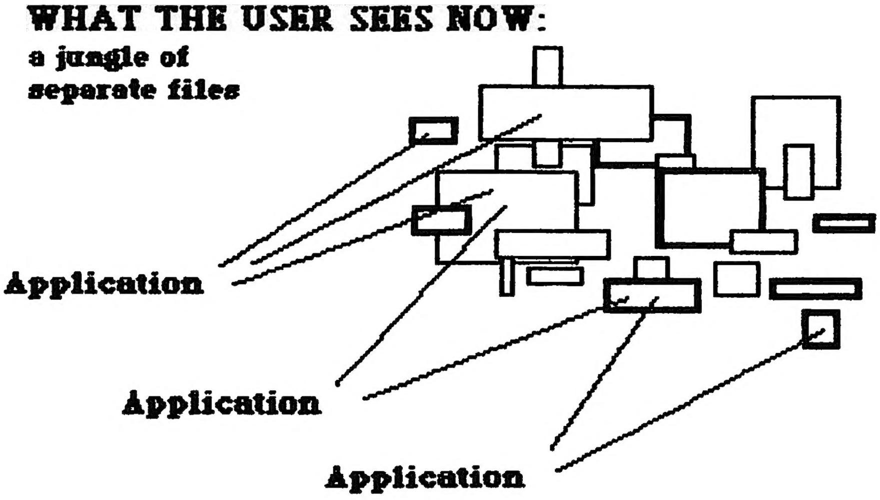
Indeed, what the world needs is a generalized form of storage that will grow and adapt and hold data for every type of application; that may be shared among all applications, yet which does not slant or gerrymander that data in any particular way, but creates a common and clarifying and universal system of storage.

THE SYSTEM OF FILES

Computer use is principally concerned with maintaining order in a chaotic jumble of files that grows ever worse.

Some files stand alone, some must be managed in clusters; but typically they are all piled together on disk with archival copies and backups in an ever-more-confusing tangle.

This is SEEZNOW.d9



INFERNAL COMPLICATIONS

Having to keep track of files is an endlessly complex and exasperating chore that may best be compared to herding mice. You have to keep pouring them from medium to medium. You have to name them all; you have to rename them; you have to keep keeping copies; you have to move them by copying and then deleting one of the survivors. (Making sure to delete the correct file in a jungle of similar names is roughly like holding and difficulty. A clean new interface to our worlds of information could sweep away a vast amount of the difficulty that people have with computers.

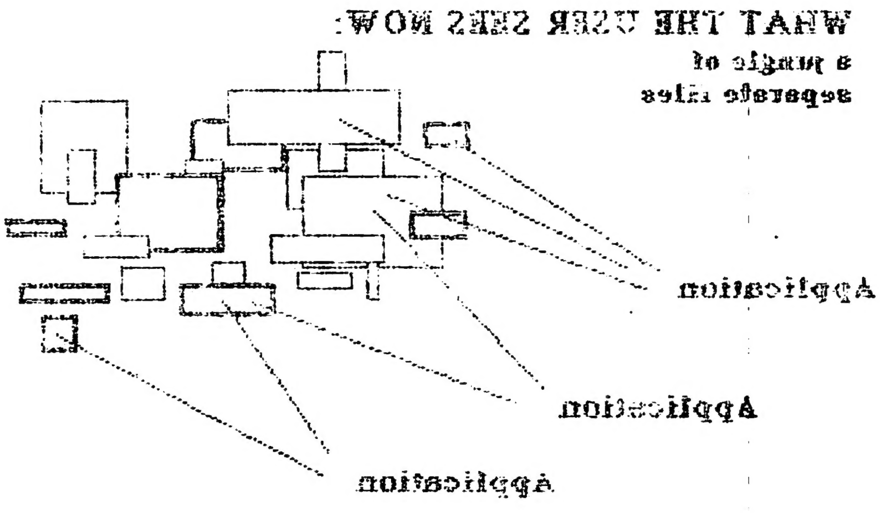
Indeed, what the world needs is a gene altered form of storage that will grow and adapt and hold data for every type of application; that may be shared among all applications, yet which does not slant or gerrymander that data in any particular way, but creates a common and unifying and universal system of storage.

THE SYSTEM OF FILES

Computer use is principally concerned with maintaining order in a chaotic jungle of files that grows ever worse.

Some files stand alone, some must be managed in clusters; but typically they are all piled together on disk with archival copies and backups in an ever-more-confusing tangle.

File is SECROW.de



INFERNAL COMPLICATIONS

Having to keep track of files is an endlessly complex and exasperating chore that may best be compared to herding mice. You have to keep pouring them from medium to medium. You have to name them all; you have to rename them; you have to keep keeping copies; you have to move them by copying and then deleting one of the survivors. (Making sure to delete the correct file in a jungle of similar names is roughly like holding your baby and your garbage over the incinerator and letting go of one, ten times a day.) Then there are the endless problems of backups, of finding space for them (often in dark crannies of disk), the escalating annoyance as disk volumes get filled to capacity and emergency transfers and deletions must be made.

Typically, from time to time, somebody just throws it all away and starts afresh, or just puts a big mess away in storage somewhere and starts over, or does a sweeping reorganization that also loses a great deal of data.

The older material can be found with difficulty as long as certain employees are still around.

THE HIERARCHY OF FILES: CATALOGS AND DIRECTORIES

Hierarchical file structures originally seemed an improvement because they gave you more places to put things, and because they had a certain fit to some applications.

Some people think hierarchically, and that's fine; but those who don't shouldn't be forced into it. (There are those who imagine that forcing your problem into a hierarchical structure promotes clear and rigorous thinking. This is, to use the politest possible term, malarkey; mapping any set of ideas to any other may present interesting exercises to the mind, but there is little point if they don't fit well.)

But unfortunately the hierarchical file model requires intricate fixed pathways we must commit to and memorize, and are very hard to change.

Yet the way we think of our work is constantly changing at the highest level. (Lucky is he whose ideas are fixed and unchanging.) Unfortunately, existing file methods stick us forever with the groupings that we start with-- whatever divisions and hierachies seemed appropriate at the beginning-- unless we do elaborate reorganizations that nobody has time for.

RELATIONS BETWEEN MATERIALS: HOW EASILY LOST

Many things need to be divided into separate files though they are connected; but the connections, not being represented, tend to evaporate. Cross-references between them; the interconnections of shared material; commonality and parallels; all these become easily lost because not easily represented.

There are many areas where we build ad-hoc programs to cope with what's not built into the your baby and your garbage over the incinerator and letting go of one, ten times a day.) Then there are the endless problems of backups, of finding space for them (often in dark crannies of disk), the escalating annoyance as disk volumes get filled to capacity and emergency transfers and deletions must be made.

Typically, from time to time, somebody just throws it all away and starts afresh, or just puts a big mess away in storage somewhere and starts over, or does a sweeping reorganization that also loses a great deal of data.

The older material can be found with difficulty as long as certain employees are still around.

THE HIERARCHY OF FILES: CATALOGS AND DIRECTORIES

Hierarchical file structures originally seemed an improvement because they gave you more places to put things, and because they had a certain fit to some applications.

Some people think hierarchically, and that's fine; but those who don't shouldn't be forced into it. (There are those who imagine that forcing your problem into a hierarchical structure promotes <u>clear and rigorous thinking</u>. This is, to use the politest possible term, malarkey; mapping any set of ideas to any other may present interesting exercises to the mind, but there is little point if they don't fit well.)

But unfortunately the hierarchical file model requires intricate fixed pathways we must commit to and memorize, and are very hard to change.

Yet the way we think of our work is constantly changing at the highest level. (Lucky is he whose ideas are fixed and unchanging.) Unfortunately, existing file methods stick us forever with the groupings that we start with-- whatever divisions and hierarchies seemed appropriate at the beginning-- unless we do elaborate reorganizations nobody has time for.

RELATIONS BETWEEN MATERIALS: HOW EASILY LOST

Many things need to be divided into separate files though they are connected; but the connections, not being represented, tend to evaporate. Cross-references between them; the interconnections of shared material; commonality and parallels; all these become easily lost because not easily represented.

There are many areas where we build ad-hoc programs to cope with what's not built into the storage system. Programs for intercomparison of files; indexing programs that mark points in text; delta-list programs to manage the history of changes; "software configuration management," which are systems for putting out different versions of the same programs from a common library. These are just a few. But I insist that if we had proper storage, all these functions would be handled in the data structure as a common reference system, available to all software.

FORMALITIES OF OPENING AND CLOSING

At the user level, most software affronts the user with tedious formalities of opening and closing that are based on this file model. Since files, their names and versions constitute the surface structure of this universe, selecting and opening these files is a level of annoyance that make firing up an application like opening a bank account. Such unnecessary distraction and formalism forbid inspiration grabbing, wear you out before you even start. This is comparable to having to deal with a desk clerk in order to make love.

(Things like Sidekick permit this to some extent, but the elaborate formalities are still necessary to put the stuff away when it's done.)

We should be able to work on numerous things at once, ping-pong style, without having to deal consciously with the formalisms of opening and closing them.

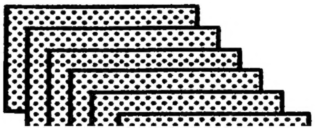
Many would say you <u>could</u> do this by conventional methods, and today's windowing packages are a start, but these are just a disguise over the existing file methods, which must be grappled with as usual when the day is done.

SUBTLER PROBLEMS

Moreover, I would argue that the conventions of files as we know them put pressure on software design to take certain oversimplified forms.

For instance, we are familiar with the "database model," in which separately coded items, or "records," may be searched on various criteria;

This is DBMODEL.D8



storage system. Programs for intercomparison of files; indexing programs that mark points in text; delta-list programs to manage the history of changes; "software configuration management," which are systems for putting out different versions of the same programs from a common library. These are just a few. But I insist that if we had proper storage, all these functions would be handled in the data structure as a common reference system, available to all software.

FORMALITIES OF OPENING AND CLOSING

At the user level, most software affronts the user with tedious formalities of opening and closing that are based on this file model. Since files, their names and versions constitute the surface structure of this universe, selecting and opening these files is a level of annoyance that make firing up an application like opening a bank account. Such unnecessary distraction and formalism forbid inspiration grabbing, wear you out before you even start. This is comparable to having to deal with a desk clerk in order to make love.

(Things like SideKick permit this to some extent, but the elaborate formalities are still necessary to put the stuff away when it's done.)

We should be able to work on numerous things at once, ping-pong style, without having to deal consciously with the formalisms of opening and closing them.

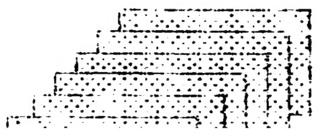
Many would say you could do this by conventional methods, and today's windowing packages are a start, but these are just a disguise over the existing file methods, which must be grappled with as usual when the day is done.

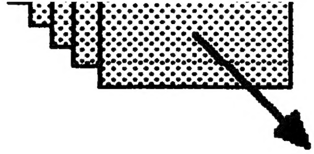
SUBTLER PROBLEMS

Moreover, I would argue that the conventions of files as we know them put pressure on software design to take certain oversimplified forms.

For instance, we are familiar with the "database model," in which separately coded items, or "records," may be searched on various criteria:

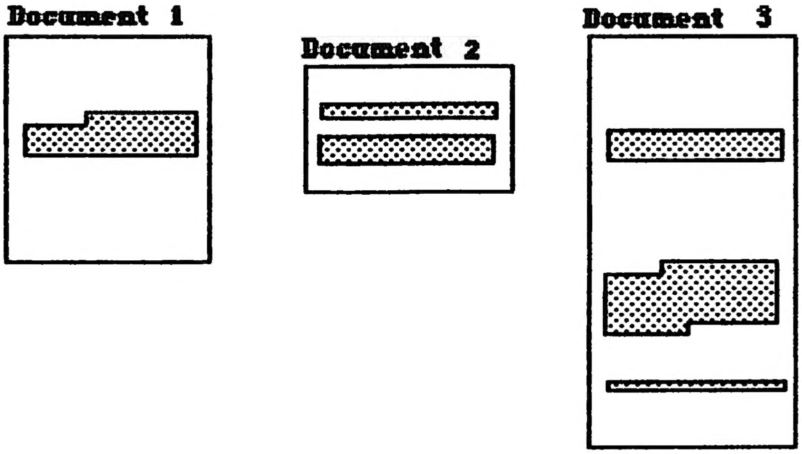
This is DBMODEL.DB





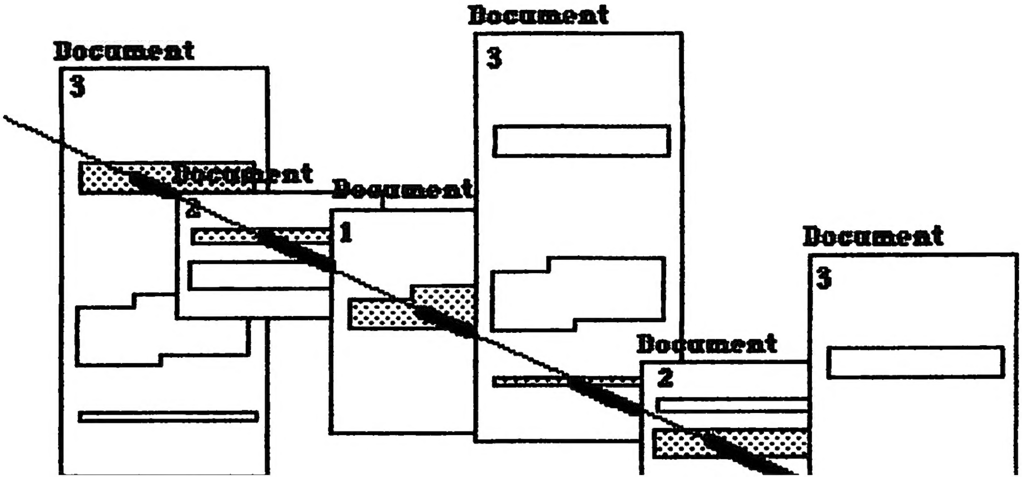
we are also familiar with the "word processing model," where sequential text may be scanned, revised and printed. But why can't these approaches be combined? What if you want to code sections of text inside their documents? But today's software doesn't let you do this. (A partial exception is a program called Dayflo.)

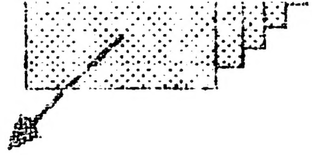
This is WPMODEL.D8



You might also like to search for the coded pieces and sort them like database items, yet see them in their living contexts at the same time.

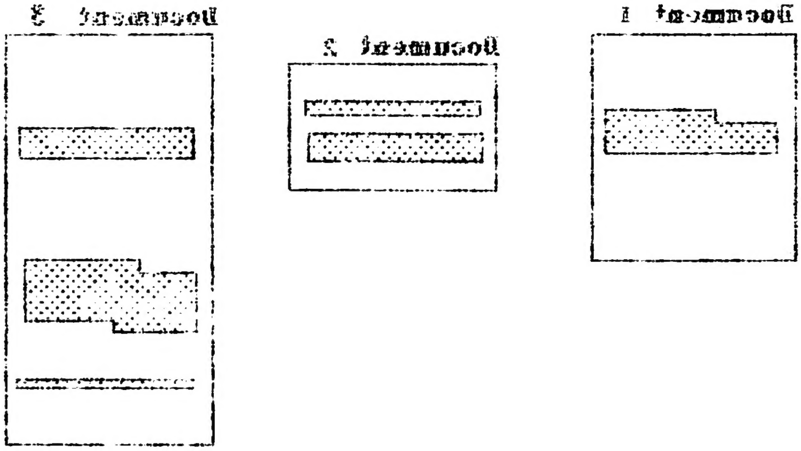
This is THECOMBO.D8





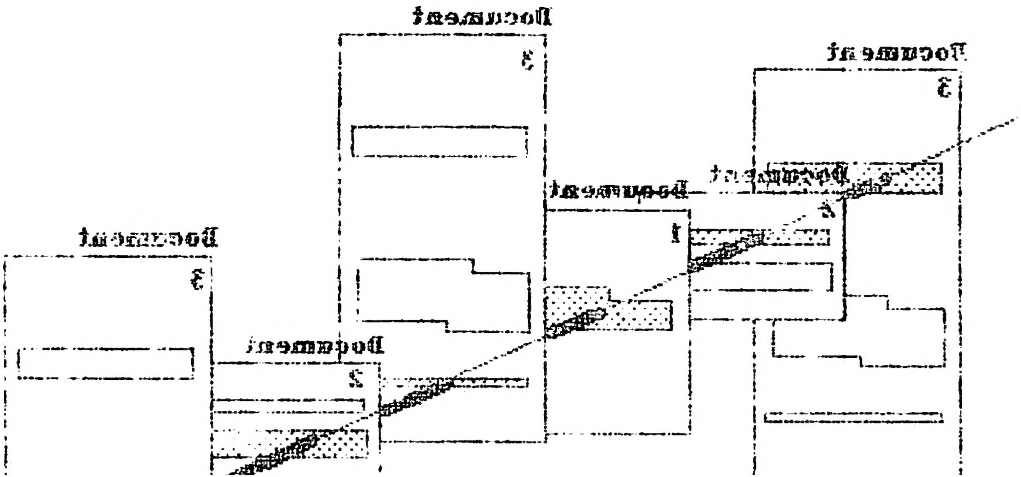
we are also familiar with the "word processing model," where sequential text may be scanned, revised and printed. But why can't these approaches be combined? What if you want to code sections of text inside their documents? But today's software doesn't let you do this. (A partial exception is a program called Dayflo.)

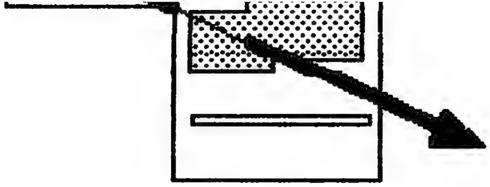
This is WPMODEL.DS



You might also like to search for the coded pieces and sort them like database items, yet see them in their living contexts at the same time.

This is THECOMPO.DS





While of course these things are "possible to program," the conventions of files strongly pressure the programmer to oversimplify both the data and the uses of the data.

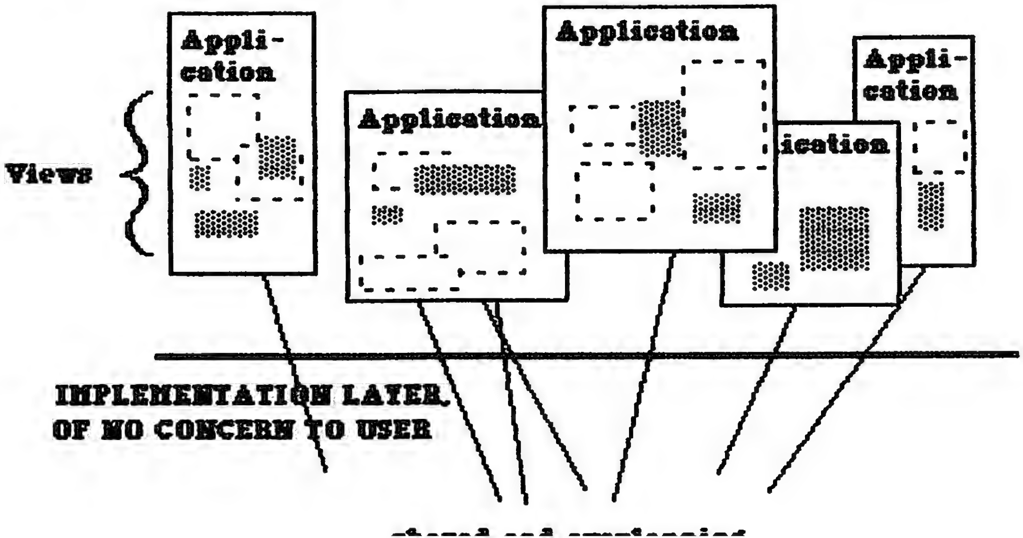
ENVISIONING A WORLD WITHOUT FILES

By a world without files, I mean a world where the user will see his or her latest work in its preferred appearance, and be able easily to trace interconnections and intercompare versions.

Each application should be like a door to a world; the user <u>opens doors</u> into applications, with multiple views of materials, each a different context or way of working on it. We need multiple pathways to the same material at the user level. Whole environments and surroundings should be easily snapshot and reopened. (This has been the intent of many integrated-software packages, notably Symphony, which tried to make possible many different views of common materials.)

This is SHUDSEE.d9

WHAT THE USER SHOULD SEE:
APPLICATIONS AND VISUALIZATIONS--
different contexts for the same materials



shared and overlapping materials, clearly demarcated as to their overlap

GRADUAL SEPARATION OF VERSIONS

The same things have be worked on in many different contexts, sometimes growing apart in different ways.

It should be possible gradually to change and separate different versions, yet keep their comonalities traced. This does not exist in any well-known system.

- - - - - - - - - - - - -

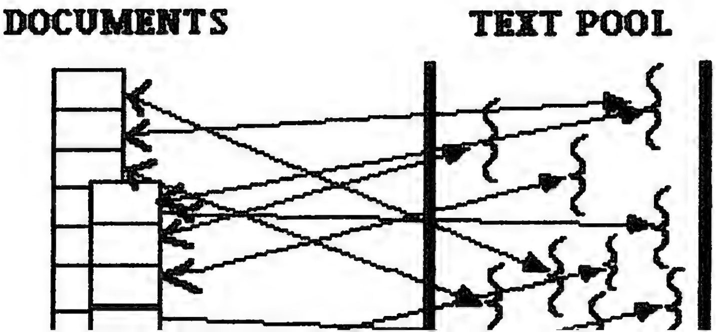
THE ALTERNATIVE

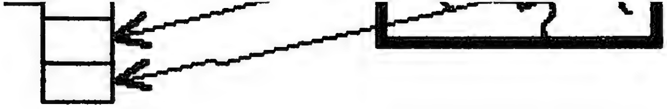
What I am proposing requires a different approach to storage. Let me explain this by degrees, starting with the case of text storage, and expand the idea toward a generalized structure.

We called the old units "files;" let us choose a new name for units that can link and overlap. I propose that we use the term "document," since text documents are often interconnected in the way we plan to permit. The <u>purpose</u> is the same as that of a file-- a useful collection of data-- but with new advantages.

Let us begin by collecting all text into a pool of dated bytes. Each byte knows when it was created. A document is a list of pointers into this pool. Conversely, each byte knows what pointers there are to it from where in which documents. (This is related to the "piece table" approach of such word processors as Samna.)

This is DOX&POOL.D8





Different documents may point into the same text pool, using the same materials in different ways. As they are modified they depart from one another, and their pointer lists change.

Given this data structure, it is plain how it may be implemented for a variety of different functions--

to go to an arbitrary part of any document;
to find what documents use a given string or byte, and where it falls in each;
to find out whether two specific documents share a piece of text;
to see the same text as it appears in two different documents;
etc.

While this has certain unusual speed advantages-- it also allows rapid jumps to arbitrary positions in text, since the program steps through pointers instead of text-- this is not its real power.

This permits a user to re-use and rework the same materials repeatedly in different ways and for different purposes, unconfused as to their origin and able to find out which sections are common between what documents and versions.

Multiple documents and users may share material, without redundant storage.

This is especially useful for "boilerplate" applications, where the same materials are reworked for different purposes.

FOR A SINGLE USER:
GREATER SPEED AND CLARITY

CLARIFICATION OF WHAT THINGS ARE
AND THEIR INTERRELATION--
NOT LIKE "FILES"

Even for one user, this approach brings clarification. The parallel maintenance of different versions, the continuing connections to the origins of each part, and differences between all working versions are easy to keep track of. Note also that the storage overhead from maintaining many documents (and saving back versions) is reduced in proportion to the amount of overlapping material.

FOR SHARED USE

Different documents may point into the same text pool, using the same materials in different ways. As they are modified they depart from one another, and their pointer lists change.

Given this data structure, it is plain how it may be implemented for a variety of different functions--

to go to an arbitrary part of any document;
to find what documents use a given
  string or byte, and where it falls in
  each;
to find out whether two specific
  documents share a piece of text;
to see the same text as it appears in
  two different documents;
etc.

While this has certain unusual speed advantages-- it also allows rapid jumps to arbitrary positions in text, since the program steps through pointers instead of text-- this is not its real power.

This permits a user to re-use and rework the same materials repeatedly in different ways and for different purposes, unconfused as to their origin and able to find out which sections are common between what documents and versions.

Multiple documents and users may share material, without redundant storage.

This is especially useful for "boilerplate" applications, where the same materials are reworked for different purposes.

FOR A SINGLE USER:
GREATER SPEED AND CLARITY

CLARIFICATION OF WHAT THINGS ARE
AND THEIR INTERRELATION--
NOT LIKE "FILES"

Even for one user, this approach brings clarification. The parallel maintenance of different versions, the continuing connections to the origins of each part, and differences between all working versions are easy to keep track of. Note also that the storage overhead from maintaining many documents (and saving back versions) is reduced in proportion to the amount of overlapping material.

FOR SHARED USE

The payoff can be even greater in a file server for many users who are sharing material. The data stay in place, they may participate in many documents of different kinds, and each use may evolve separately-- but connections may be continually traced among them. Where the same materials and boilerplate are repatedly used, as in law offices, such facilities can be fital.

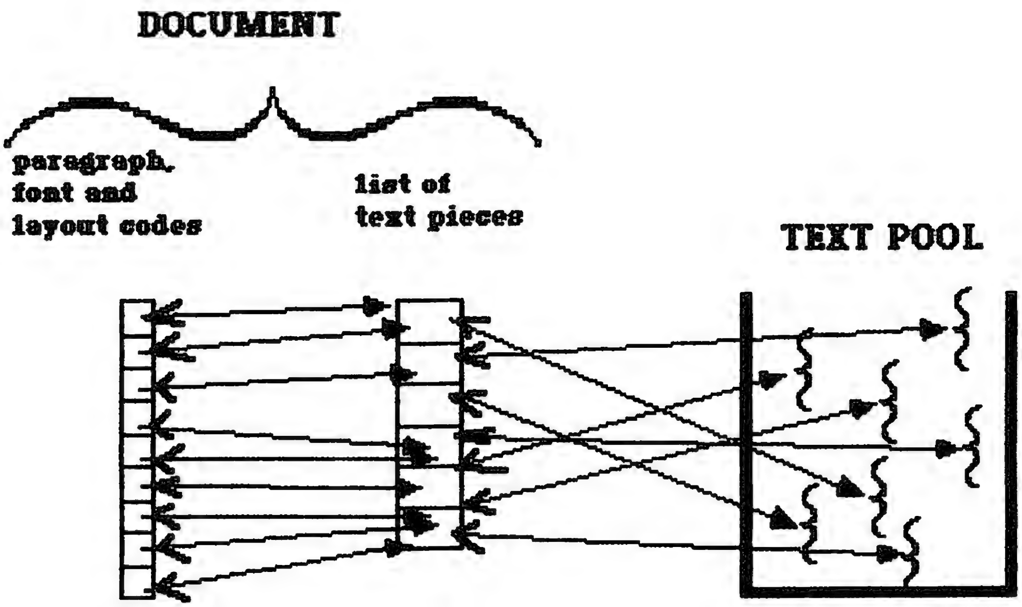
FORMATTING

The text pool should contain only "pure text," uncluttered with information about paragraphs, fonts, etc., that will be different among different users. Thus formatting information must be purged from the text, in order to assure that only a clean base of sharable materials be in the common pool.

Therefore a second set of pointers is needed, isolating the formatting information.

Formats are now sets of poiners into this pool, with separate format codes.

This is FORMATNG.D8



GENERALIZING TO ALL FORMS OF DATA

This same approach may be deeply generalized. What we have found is a way to mark the data from outside, so that these applications which want to share the data and the markers may do so freely, but the markers do not clutter the data for applications where they're not wanted.

We really want a much more general facility, one which permits arbitrary markings in a pool of data which may be used for different purposes by

The payoff can be ever greater in a file server for many users who are sharing material. The data stay in place, they may participate in many documents of different kinds, and each use may evolve separately-- but connections may be continually traced among them. Where the same materials and boilerplate are repeatedly used, as in law offices, such facilities can be vital.

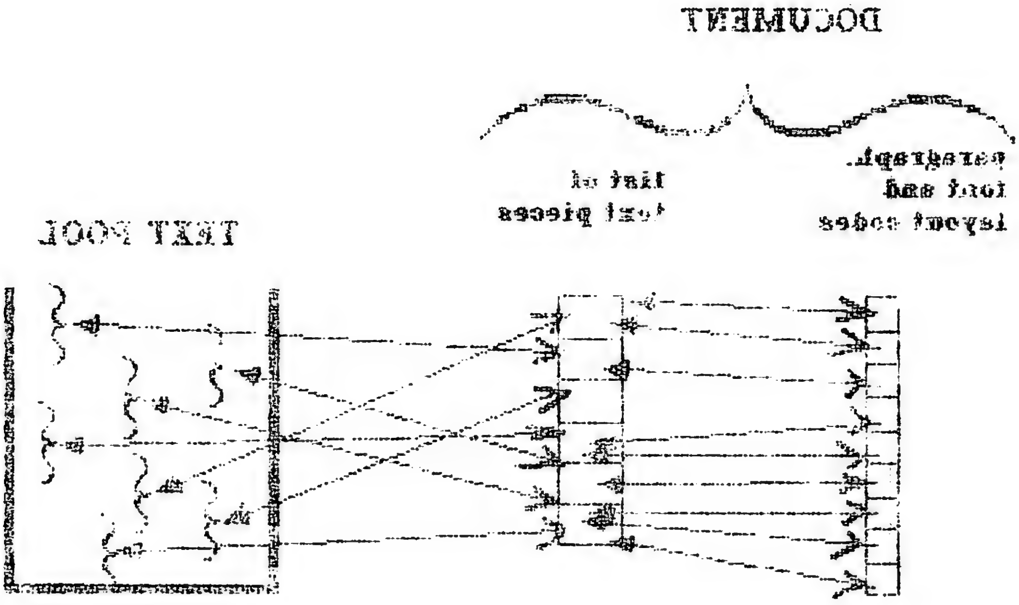
FORMATTING

The text pool should contain only 'pure text,' uncluttered with information about paragraphs, fonts, etc., that will be different among different users. Thus formatting information must be purged from the text, in order to assure that only a clean base of sharable materials be in the common pool.

Therefore a second set of pointers is needed, isolating the formatting information.

Formats are now sets of poiners into this pool, with separate format codes.

This is FORMATNG.DS



GENERALIZING TO ALL FORMS OF DATA

This same approach may be deeply generalized. What we have found is a way to mark the data from outside, so that these applications which want to share the data and the markers may do so freely, but the markers do not clutter the data for applications where they're not wanted.

We really want a much more general facility, one which permits arbitrary markings in a pool of data which may be used for different purposes by different users of these data, but which do not obstruct or encumber the data for users who don't want them.

We may think of this as a generalization of this format coding system.

- - - - - - - - - - - -

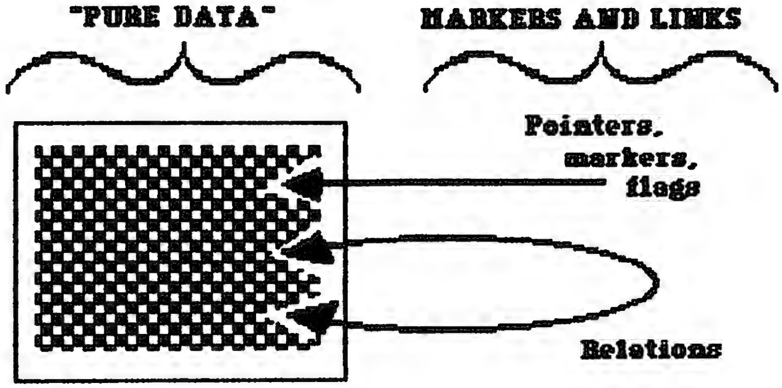
MARKERS AND LINKS

We want to mark and link data for many purposes. You may use <u>markers </u>to point at specific items, to hold your place, to indicate items or sections of a certain type.

You may also want <u>links</u> between different parts of your data-- to show comments; to show structural interconnections; to show corresponding parts (e.g. between code and documentation); and so on.

We want to move markers and links out of the data, but keep them where different users can use them for different purposes. All the different types of links and markers are to be kept in a separate pool.

This is DATALNKS.D10



These markers and links may be of many types, but by pooling them we gain different users of these data, but which do not obstruct or encumber the data for users who don't want them.

We may think of this as a generalization of this format coding system.

- - - - - - - - - - -

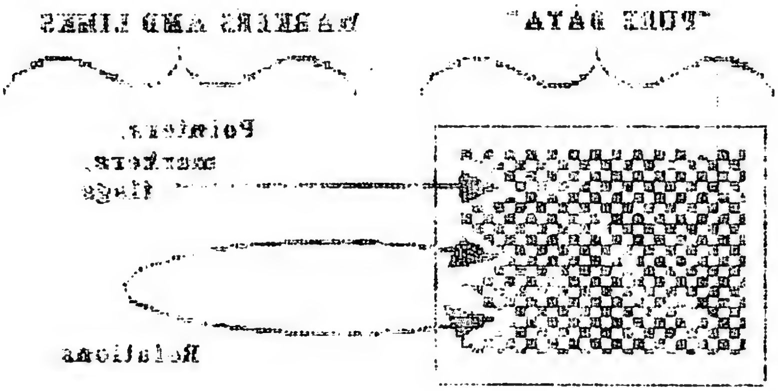
MARKERS AND LINKS

We want to mark and link data for many purposes. You may use markers to point at specific items, to hold your place, to indicate items or sections of a certain type.

You may also want links between different parts of your data-- to show comments; to show structural interconnections; to show corresponding parts (e.g. between code and documentation); and so on.

We want to move markers and links out of the data, but keep them where different users can use them for different purposes. All the different types of links and markers are to be kept in a separate pool.

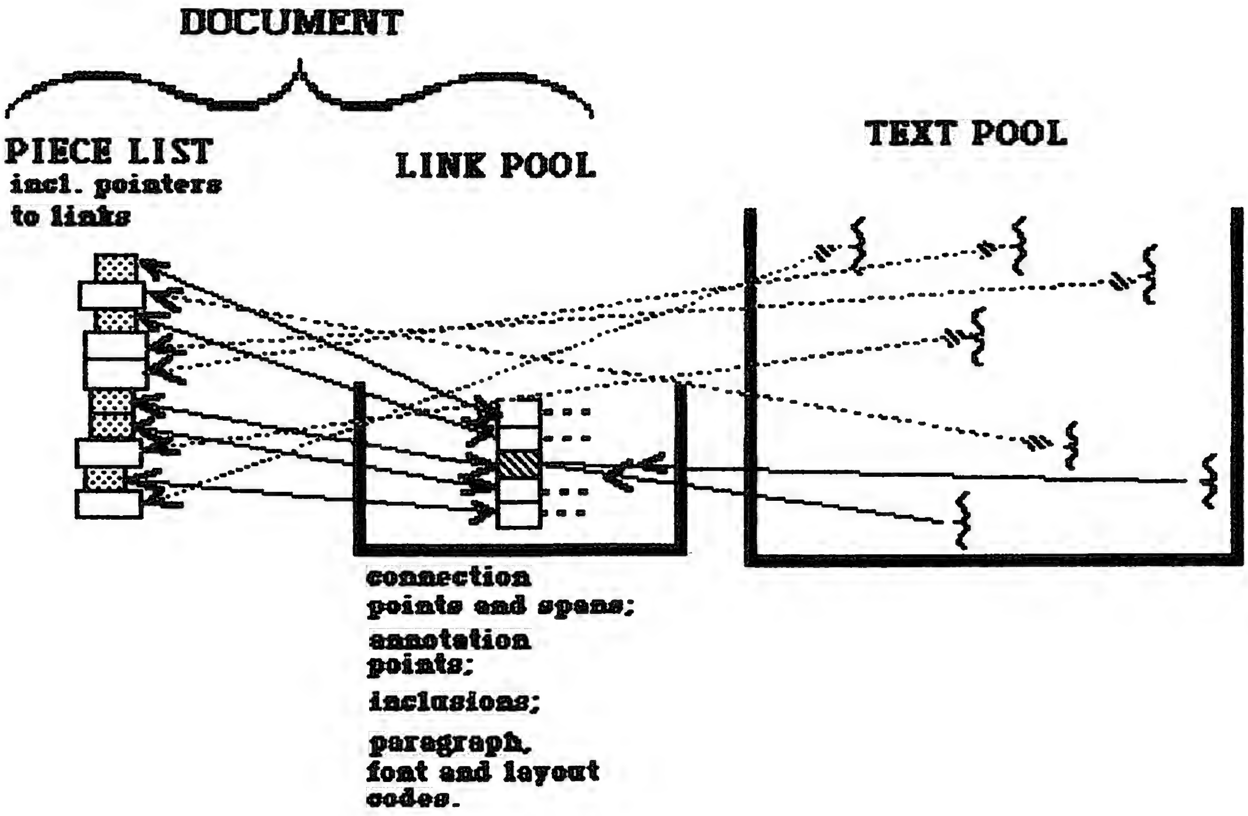
This is DATALNKS.D10



These markers and links may be of many types, but by pooling them we gain numerous advantages: we can find which ones attach to any given section of data; we can search them by type, by time of entry, by owner and so on.

Let's consider how this will work in a text system-- a text system for the storage and maintenance of linked materials (hypertext) and arbitrary forms of annotation.

This is LINKPOOL.D10



Each document can be read in sequence; every use of a piece can be traced to the other documents it has migrated to; and overlapping pieces can be coded in numerous ways-- without obstructing uses that don't need them.

TRICKY APPLICATIONS

What we call here "links" are easily used for database-type coding. Thus that this data structure easily supports the application described earlier-- coding parts of documents and searching for them so they may be seen in their living document contexts. (Neither word

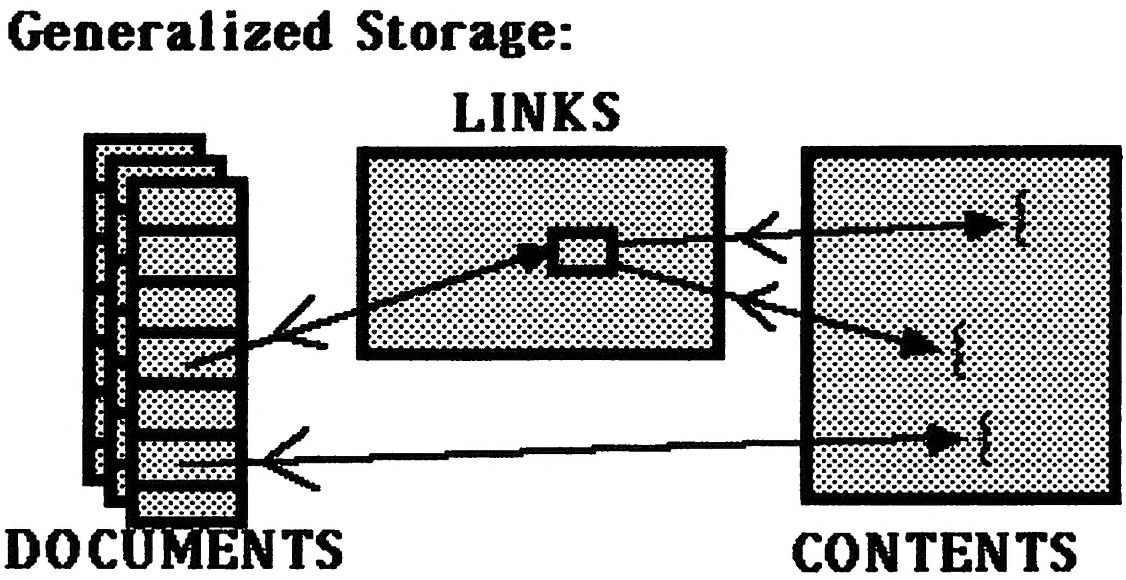
processing nor database has advantage to the programmer over their combination.)

Indeed, this style of data structure exerts no pressure to design applications a given way; and so escapes the styles of problem analysis, and divisions into simplified solutions, that are fostered by conventional files.

GENERALIZED STORAGE

This notion generalizes to a new storage paradigm with wide-ranging implications. I believe it is a unifcation which can exactly represent the intrinsic structure of all data.

This is GNRLSTOR.D8



Though it is not generally recognized, this is needed in ALL FIELDS AND APPLICATIONS. Engineering, law, medicine, computer science, art history, entomology and intelligence work have the same problems of representing linkage and origins of data, commonality between documents, historical backtrack. This is true for all types of data-- text, graphics, business data, scientific data.

OFFICE AUTOMATION

And there is no way to have the automated office without it. Without these facilities, I submit, there is no way to build the kinds of features that the true paperless office will require, eliminating the debris of loose and lost files that are accumulating everywhere.

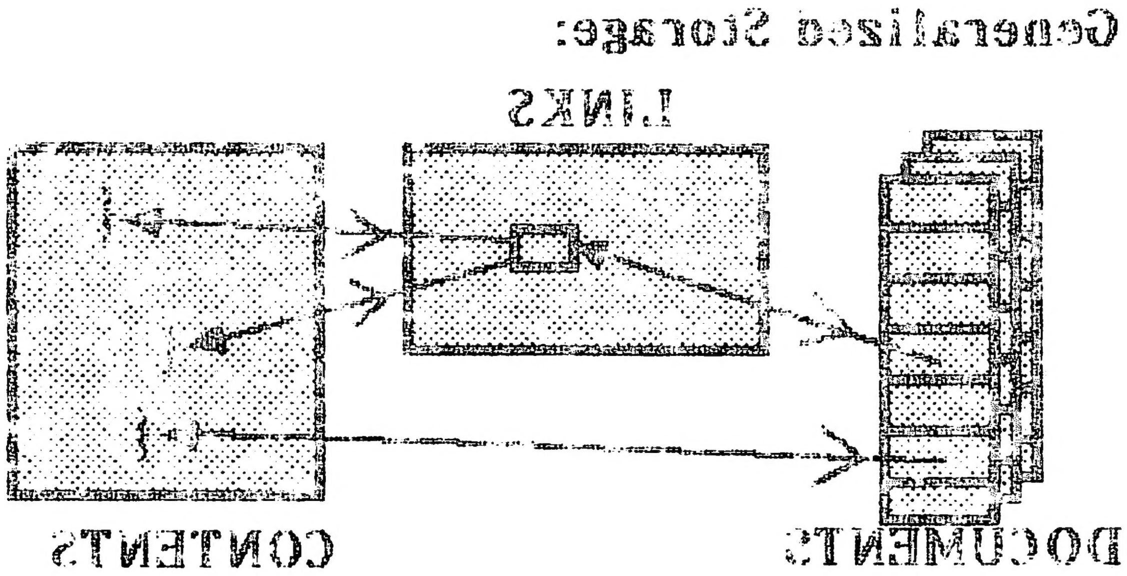
processing nor database has advantage to the programmer over their combination.)

Indeed, this style of data structure exerts no pressure to design applications a given way; and so escapes the styles of problem analysis, and divisions into simplified solutions, that are fostered by conventional files.

GENERALIZED STORAGE

This notion generalizes to a new storage paradigm with wide-ranging implications. I believe it is a unification which can exactly represent the intrinsic structure of all data.

This is GENLSTOR.DB



Though it is not generally recognized, this is needed in ALL FIELDS AND APPLICATIONS. Engineering, law, medicine, computer science, art history, entomology and intelligence work have the same problems of representing linkage and origins of data, commonality between documents, historical backtrack. This is true for all types of data-- text, graphics, business data, scientific data.

OFFICE AUTOMATION

And there is no way to have the automated office without it. Without these facilities, I submit, there is no way to build the kinds of features that the true paperless office will require, eliminating the debris of loose and lost files that are accumulating everywhere.

PUBLISHING

On-line publishing so far has sold chunks of text that are too big. Users need to be able to browse on-line through forests of interconnected material, paying for small pieces as they go. The approach we have described is excellent for on-line publication with royalty, since everything's origin is identifiable down to the byte level.

When a document is read out by a user, the owner of each byte may be minutely rewarded from the user's account with no intricate mechanism-- just as the user of a jukebox automatically pays royalties to a song's owner and performer.

We may even envision A NEW LITERATURE-- where linkage, intrinsic everywhere, becomes now a part of the structure of the writing itself.

ARCHIVING

The problem of digital archives is growing at an extraordinary pace. There is an increasing chaos of different programs and formats.

It is not known whether the software used to produce some of these data will even continue to exist, let alone be maintained, when historians want to study the material-- let alone next year, when the boss wants to find out what happened.

There are word processors and spreadsheets of every conceivable kind, there are forests of graphics and 3D shapes produced by a variety of systems, and much more. The increasing need for archival storage demands that a universal archival form be found-- one to which all existing data structures and arrangements may be mapped.

What we need is a stable and generalized form of storage on which persons of good will can agree, leaving out nothing which is represented in any other system. And I believe this can be achieved.

EXPANSION

Separate installations have their limits. We need a stable indexing scheme that works across node boundaries, if we are to have an indefinitely expansible network OF instantaneous accessibility. And that is what must be.

PUBLISHING

On-line publishing so far has sold chunks of text that are too big. Users need to be able to browse on-line through forests of interconnected material, paying for small pieces as they go. The approach we have described is excellent for on-line publication with royalty, since everything's origin is identifiable down to the byte level.

When a document is read out by a user, the owner of each byte may be minutely rewarded from the user's account with no intricate mechanism-- just as the user of a jukebox automatically pays royalties to a song's owner and performer.

We may even envision A NEW LITERATURE-- where linkage, intrinsic everywhere, becomes now a part of the structure of the writing itself.

ARCHIVING

The problem of digital archives is growing at an extraordinary pace. There is an increasing chaos of different programs and formats.

It is not known whether the software used to produce some of these data will even continue to exist, let alone be maintained, when historians want to study the material-- let alone next year, when the boss wants to find out what happened.

There are word processors and spreadsheets of every conceivable kind, there are forests of graphics and 3D shapes produced by a variety of systems, and much more. The increasing need for archival storage demands that a universal archival form be found-- one to which all existing data structures and arrangements may be mapped.

What we need is a stable and generalized form of storage on which persons of good will can agree, leaving out nothing which is represented in any other system. And I believe this can be achieved.

EXPANSION

Separate installations have their limits. We need a stable indexing scheme that works across node boundaries, if we are to have an indefinitely expansible network OF instantaneous accessibility. And that is what must be.